

*Department of Electrical & Systems Engineering*

*Departmental Papers (ESE)*

---

*University of Pennsylvania*

*Year 2005*

---

Combined Software/Hardware  
Implementation of a Filterbank  
Front-End for Speech Recognition

Athanasios Mouchtaris\*      Yuan Cao<sup>†</sup>      Shehzad Khan<sup>‡</sup>  
Jan Van der Spiegel\*\*      Paul Mueller<sup>††</sup>

\*University of Crete

<sup>†</sup>University of Pennsylvania

<sup>‡</sup>University of Pennsylvania

\*\*University of Pennsylvania, jan@seas.upenn.edu

<sup>††</sup>Corticon, Inc.

Copyright 2005 IEEE. To be published in the *Proceedings of the IEEE 2005 Workshop on Signal Processing Systems (SIPS 2005)*, pages 436-441.

Conference URL: <http://www.vlsi.ee.upatras.gr/sips2005/>

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Pennsylvania's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to [pubs-permissions@ieee.org](mailto:pubs-permissions@ieee.org). By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

This paper is posted at ScholarlyCommons.

[http://repository.upenn.edu/ese\\_papers/143](http://repository.upenn.edu/ese_papers/143)

# Combined Software/Hardware Implementation of a Filterbank Front-End for Speech Recognition

A. Mouchtaris  
Department of Computer Science  
University of Crete  
Heraklion, Crete 71409  
Greece

Y. Cao, S. Khan, J. Van der Spiegel  
Electrical and Systems Engineering  
University of Pennsylvania  
Philadelphia, PA 19104  
USA

P. Mueller  
Corticon Inc.  
King of Prussia, PA 19406  
USA

**Abstract**—In this paper, a cost-effective implementation of a programmable filterbank front-end for speech recognition is presented. The objective has been to design a real-time bandpass filtering system with a filterbank of 16 filters, with analog audio input and analog output. The output consists of 16 analog signals, which are the envelopes of the filter outputs of the audio signal. These analog signals are then led to an analog neural computer, which performs the feature-based recognition task. One of the main objectives has been to allow the user to easily change the filter specifications without affecting the remaining system, thus a software implementation of the filterbank was preferred. In addition, the neural computer requires analog input. Therefore, we implemented the filterbank on a PC, with the input A/D and the output D/A performed by the PC stereo soundcard. Since multiple analog outputs are necessary for the neural computer (one for each filter), it then follows that the soundcard output should contain the multiplexed 16 filter outputs, while a hardware module is needed for demultiplexing the soundcard output into the final 16 analog signals.

## I. INTRODUCTION

We present in this paper a cost-effective implementation of filterbank as a front-end for a speech recognition system. This speech recognition system is based on a biologically-inspired system proposed by Mueller *et. al* [1]. There, an auditory-based filterbank is proposed as a front-end to a phoneme recognition system that is implemented on an analog neural network (neural computer). The main part of this front-end is a bank of Bark-scaled filters that resemble the processing of sounds by the human cochlea. The filter outputs are then processed by an envelope detector, and these signals must then be led to the neural computer in analog form. The neural computer performs the recognition task by extracting several features from these signals. These features are based on the absolute energy in each band and across different bands; on the energy slope in each band; on energy durations; on the onset/offset durations; and on energy slope correlations across adjacent bands. The final result of the system is recognition of the spoken phonemes. An advantage of the analog neural computer is its real-time recognition performance, as well as programming flexibility. Our goal has been to design the filterbank front-end, which includes 16 Bark-scaled filters covering the frequency range 0-4 kHz, and envelope detectors for each filter output.

The fact that the front-end proposed here is designed for the particular recognition system of [1], does not restrict its use for other similar recognition systems. Ali *et. al* [2], [3], [4], have proposed an auditory-based front-end for acoustic-phonetic speech recognition (referred to as Average Localized Synchrony Detector or ALSD). The ALSD front-end also includes bandpass filters and envelope detectors, but additional processing of the speech signal is needed. For such cases, these additional functions can be easily added to our design, since our implementation of the front-end is mostly done on software, as explained next.

Our goal has been to implement the filtering and A/D and D/A conversion on a PC. The ADC/DAC can be performed by the PC soundcard, while the filtering task can be implemented in software, processing the soundcard input and producing the output in real-time. For the software part, we decided to use Matlab, which offers the advantage of simple real-time data processing due to the Data Acquisition Toolbox (DAQ). In addition, Graphical User Interface (GUI) design, as well as signal processing functions (such as filter design and real-time filtering) can be easily implemented. As already mentioned, the final task – the actual speech recognition – is performed on an analog neural computer, which is designed to accept as input the 16 filter outputs (after the envelope detector) in analog form. We could have used a multichannel soundcard for providing these 16 analog channels to the neural computer from the PC. However, the 16 filter outputs are low-frequency signals and consequently do not require high sampling rates for the D/A conversion. Thus, using a multichannel soundcard with 16 analog outputs would be an easy to implement but costly option. Furthermore, it would be impractical to follow this approach if more than 16 channels were used (which is possible in future versions of our system). We used a stereo soundcard, which automatically means that a multiplexing/demultiplexing scheme is needed, so that we can produce 16 analog outputs from the 2 analog outputs of the soundcard. In our designed scheme, the multiplexing is performed in the software module, while the hardware module is responsible for demultiplexing. This added complexity results in a cost-effective system, which can be easily generalized if more than 16 channels are needed. The system design is detailed in the following sections.

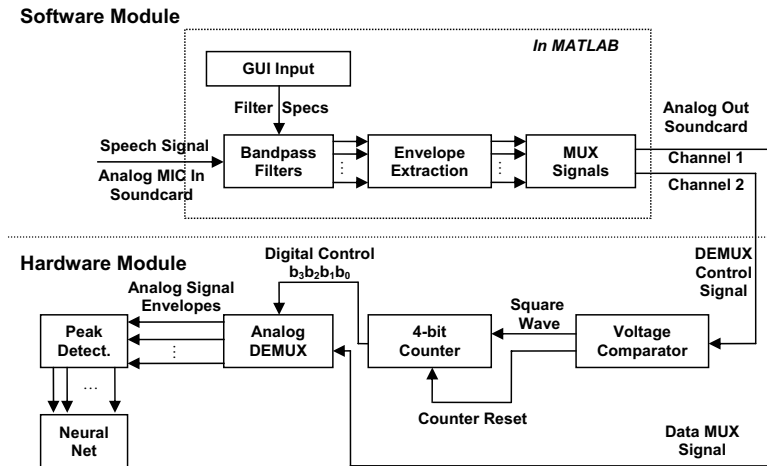


Fig. 1. Block diagram of the designed front-end. It consists of the software part (in Matlab), which is responsible for the bandpass filter design and the real-time filtering, as well as the envelope extraction and signal multiplexing. The hardware part is needed for demultiplexing the analog multiplexed signal into the various filter outputs, which are then loaded to the neural computer.

## II. SYSTEM DESIGN

In Fig. 1 the block diagram of the designed system is depicted. There are two modules, the software and hardware modules. The software module has been designed for implementing the bandpass filtering and envelope detectors, as well as the multiplexing of the produced envelopes, while the hardware module performs the analog demultiplexing (switching). More details are given next.

### A. Software Module

Initially, the speech is acquired by a microphone through the soundcard that performs the A/D conversion. The software part is implemented in Matlab. The DAQ Toolbox allows for Matlab programs to communicate with the soundcard buffers, thus the sampled speech signal (8 kHz sampling rate was used) is readily available for processing. The GUI we designed (also using Matlab GUI commands), enables the user to specify the filterbank edge frequencies (*i.e.* cutoff frequencies for each filter), the sidelobe filter attenuation, and a different gain for each filter. The filter design is based on the FIR Kaiser filter design method [5], while it holds that any other method, FIR or IIR, could be used for obtaining the filterbank. An example of a filterbank we designed is given in Fig. 2, where the sidelobe attenuation is 40 dB, the gain for all filters is 1, and the edge frequencies are given in Fig. 3, where the Graphical User Interface (GUI) of the program is shown. In Fig. 3 it can also be seen that the user can specify one of the filter outputs (before the envelope detector) to be viewed in real-time on a Matlab window.

In order to do the filtering in real-time, the speech signal is processed in non-overlapping segments. The soundcard is responsible for buffering the signal, while the size of the buffer can be specified in Matlab. The size of the buffer is a very important parameter, since the total delay of the software

module is double the size of the buffer (in samples). This can be better understood by considering first the case of no processing, *i.e.* when the speech is simply loaded in the input buffer and then sent to the output buffer of the soundcard. For this simple system, the time needed for the first speech sample to be loaded in the input buffer until the time it is sent to the output buffer (and thus the soundcard output), equals the buffer size in samples. This is true since the input buffer must be loaded completely until it is sent to the output buffer. Thus the system delay in the simple A/D followed by D/A system equals the buffer size. This is the delay introduced by the soundcard, if we consider the delays introduced by the ADC/DAC and the operating system negligible. If the signal must be processed as well, before it is sent to the output, then additional time is needed. This processing must be completed in an additional period of time that equals the buffer size, so that no speech data are lost during acquisition. This can be explained since this is the time that is needed for the the next segment of speech data to be loaded into the input buffer of the soundcard. If more time is needed for processing than that of a buffer size, a third segment will be loaded, while none of the first two segments will be sent to the output. However, the two segments in the input and output buffers must come from consecutive segments of the speech signal or else data will be inevitably lost.

Since our system processes the speech signal in non-overlapping segments, a block convolution procedure for filtering each segment separately is used. This is important so that the block convolution output will be exactly equal to the signal that would be produced if the whole speech signal was filtered at once. The filtering is done using the convolution command of Matlab, since FIR filters are used in our implementation. In this case, the output of each filtered segment will be of size  $length(filterOut) = length(buffer) + length(filter) - 1$ .

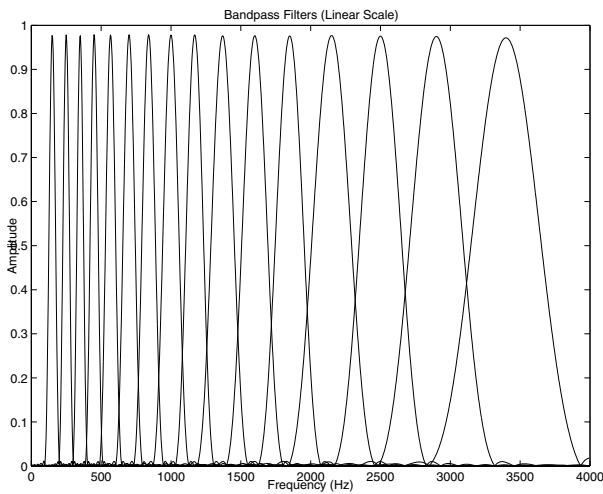


Fig. 2. Magnitude of the designed bandpass filters in the frequency domain. The parameters for the filter design are defined by the user using the GUI. An example parameter setting for the GUI is shown in Fig. 3, and these correspond to the filters depicted in this figure.

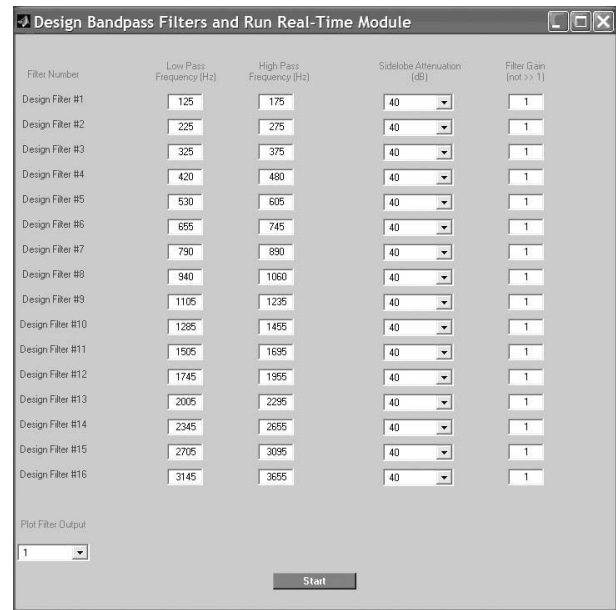


Fig. 3. An example of the GUI design, for a particular choice of filter parameters. The user can specify the filter edge frequencies, sidelobe attenuation and filter gain, as well as which filter output should be displayed (before the signals are processed by the envelope detectors).

In order to produce the correct result, *i.e.* the result of the convolution of the whole speech signal with each bandpass filter, we employ an overlap-add block convolution procedure [5]. For each segment that has been filtered, the output buffer must contain the first  $length(buffer)$  samples of the filter output, but to these we must also add the last  $length(filterOut) - length(buffer) = length(filter) - 1$  samples of filter output of the previous segment. These are added to the first  $length(filter) - 1$  samples of the total  $length(buffer)$  of the current segment. The result of this procedure will be exactly equal to the convolution of the whole speech signal with the FIR filter.

Following the filtering, 16 signals are produced and each one is passed through a half-wave rectifier and a low-pass filter with cutoff frequency of 50 Hz (envelope detector part). The 16 derived envelopes are then downsampled by a factor of 16, so that the multiplexed signal that is designed at the next step of the software module will have the same dimensionality as the input sampled speech signal. Note that since the low-pass filter used for the envelope detector has a 50 Hz cutoff frequency and the sampling rate is 8 kHz, downsampling by a factor of 16 will still result in oversampled versions of the filter envelopes. This is true since after downsampling the new sampling rate will be  $8000/16 = 500$  Hz, while for each signal envelope 100 Hz would be adequate (50 Hz maximum frequency). In other words, no significant information is lost due to the downsampling.

Multiplexing is the last task that occurs at the software module. The objective is to place the 16 downsampled envelopes in a single signal, which will then be led to the soundcard output and will occupy one of the two soundcard analog channels. The demultiplexing is performed in the hardware module based on a synchronization signal that is designed in the software module and is described in the next paragraph.

One restriction for multiplexing is that there must be a minimal delay between the envelope signals after demultiplexing. In the ideal case, all the 16 signals should be available concurrently after demultiplexing, but this is not possible in practice since the demultiplexer switches from one channel to the next in a serial fashion (more details about the demultiplexing procedure can be found in the following sub-section, describing the hardware module).

In order to minimize the delay between the signals after demultiplexing, we limit the duration of each channel during multiplexing in the software module. In other words, for each signal that is sent to the output buffer, we allocate a very small number of samples to each of the 16 channels during multiplexing (in essence concatenating the 16 different signals). Thus, we do not divide the output buffer in 16 segments, but rather in a large number of segments, each containing few samples of each channel; then, the channels in the buffer are repeated in a circular fashion. This can be easier visualized in the extreme case when the multiplexed signal is created by simply concatenating the different envelopes at each buffer. For example, with a buffer of 2048 samples (thus a system delay of 4096 samples or approximately 0.5 sec.), each of the 16 filter envelopes will consist of  $2048/16 = 128$  samples. Note that we prefer buffer sizes that are powers of 2 for compatibility with Matlab buffering restrictions. The upper part of Fig. 4 corresponds to this multiplexing scheme. During demultiplexing, 15 out of the 16 output channels will be inactive for the time it takes to demultiplex one channel. In this case, this time will be  $128/8000 = 16$  msec. Similarly, the time that each channel remains inactive (*i.e.* one cycle until all

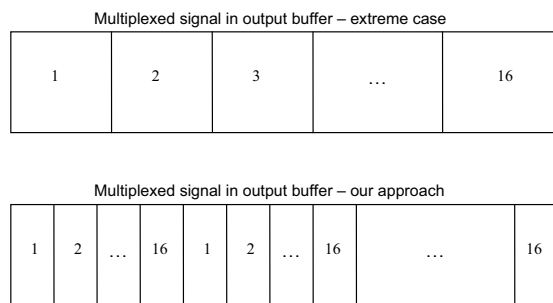


Fig. 4. Two possible choices for the multiplexing scheme. In the upper part, each of the 16 channels occupies one part of the output buffer only. In the bottom part, which is the one employed, each channel occupies a certain number of samples, so that each channel is repeated more than once for each buffer.

other channels are demultiplexed) will be  $15 \cdot 128 / 8000 = 240$  msec. This is a relatively long period of time, and in fact there is the restriction in the design of the neural computer module that each channel should not remain inactive for more than 10 msec. With this last specification, for the given sampling rate we restrict the duration of each channel during multiplexing to 4 samples, which corresponds to  $15 \cdot 4 / 8000 = 7.5$  msec. for each channel cycle, which is within the specifications (*i.e.* less than 10 msec.). Note that this is a value that does not depend on the buffer size but only on the sampling rate and the number of filters. In this case, each of the 16 channels will last for  $4 / 8000 = 500 \mu\text{sec}$  per cycle. The multiplexed signal structure for this case is given in the bottom part of Fig. 4.

In order to correctly demultiplex the designed signal, another synchronization signal is needed that will control the switching in the hardware module. Since we have only used so far one of the available 2 channels of the soundcard for the multiplexed signal, the other channel can be used for the synchronization signal. This signal is created so that it takes value of 1 for half of the duration of each channel and a value of 0 for the other half, *e.g.* two samples equal to 1 and two samples equal to 0, as in Fig. 5. As can be seen in the figure, this is repeated for all the 16 channels, for the duration of the multiplexed signal. Note that the amplitude for the first channel is double the amplitude of the others, *i.e.* 2 for the first two samples and 0 for the remaining two. This is done so that later, during the hardware implementation of the demultiplexing, there is an indication as to not only which part of the signal corresponds to which one of the 16 channels, but also of the actual label of the channel. In other words, exactly which of the 16 channels is active at a particular time during demultiplexing must be known. This is important since each channel corresponds to a particular filter and thus a specific frequency band. Apparently, the synchronization signal we designed will take a sinc-like form after it is passed through the D/A converter of the soundcard, which is an issue that is addressed in the following sub-section.

To conclude, the final output of the the software module consists of two analog signals that can be obtained from

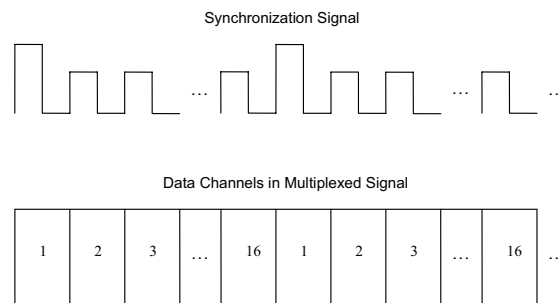


Fig. 5. Design of the synchronization (control) signal during multiplexing. For each of the 16 channels, a pulse is switched from high to low at the middle point of the channel. For the first channel, the pulse has double amplitude than in the remaining channels, so that during demultiplexing each channel can be correctly labeled.

the 2 channels of the soundcard. One channel contains the multiplexed signal of the 16 filter envelopes, while the second channel contains the synchronization signal that will be used for controlling the analog demultiplexer in the hardware module. This module is described next.

### B. Hardware Module

The various components of the hardware module can be seen in Fig. 1. The hardware components required are very inexpensive and easily available. The analog multiplexed signal is led directly to the analog demultiplexer (possibly after it is amplified using *e.g.* an operational amplifier or op-amp). On the other hand, the analog synchronization signal is led to a digital voltage comparator and then to a 4-bit counter. The reason is that the analog demultiplexer (we used a 16-1 analog multiplexer/demultiplexer) requires a digital control signal for switching between the 16 channels. In other words, the analog demultiplexer must be controlled by 4 different digital logic signals, so that switching can be controlled accordingly (for example channel 1 will correspond to all 4 control signals being low).

The role of the voltage comparator is to digitize the synchronization analog signal. This signal was created in the software module so that it is 1 for half of the duration of each channel, and 0 for the remaining part (and double amplitude for channel 1, the first of the 16 channels). However, this signal after it is passed through the D/A converter of the sound card, and since the sampling rate is 8 kHz, attains a sinc-like form. This is similar to the case when a rectangular pulse is filtered with a low-pass filter. The voltage comparator is then used to convert this analog sinc-like signal to the square wave that it was originally designed to be. Additionally, a second voltage comparator is used with double reference voltage than the voltage used for the first comparator. This second comparator is used for detecting the presence of the pulse for channel 1, which has double the amplitude compared to the other pulses. This signal then is used as a reset to the counter, as explained in the next paragraph. It should be mentioned at this point that an important issue in practice is to implement a comparator

with *hysteresis* [6]. In essence, this approach includes the addition of a feedback circuit to the comparator, where a small amount of the digital output is added back to the comparator input. We found that hysteresis offered great advantages for the robustness of the circuit, since the comparator output is led into a counter which has proved to be very sensitive to noise and jitter produced by the comparator. The addition of the hysteresis circuit addressed this noise problem.

The output of the first comparator is then led to the input of a sequential 4-bit counter. The counter is set for counting the rising edges in the square wave, so it changes value at the beginning of each channel. Since we use a 4-bit counter, the counter output will produce 4 digital outputs, corresponding to each one of the bits (from the less significant bit - LSB - to the most significant bit - MSB). The use of the second comparator as a reset signal to the counter will guarantee that the counter output 0000 will correspond to channel 1 (*i.e.* the only channel that gives output 1 for the second comparator, as explained in the previous paragraph). At the same time, the remaining channels will be labeled correctly as well, since they have been multiplexed sequentially (*i.e.* channel 2 will correspond to counter output 0001, *etc.*). Finally, these 4 digital outputs of the counter will be led to the control inputs of the demultiplexer. The 16 demultiplexed signals are then passed through an analog peak detector (information about possible implementations of the analog peak detector can be found in [6]). An optional step which we implemented has been the addition of BAR-Led's (with the use of an appropriate digital controller). This is a useful step that enables visualization of the energy at each band in real-time. Note, though, that Matlab can also be used for plotting various data, in the form that they have before the D/A conversion. As it was mentioned earlier in this section, the GUI we designed offers such functionality.

### III. MEASUREMENT RESULTS

It is important to mention that the delay introduced to the signals by the hardware part is much smaller than that of the software part, and is in the order of  $\mu\text{sec}$ . On the other hand, the delay introduced by the software part depends on the processing speed of the PC used and as explained earlier is double the duration of the input buffer. This value is chosen by the user, however there is a minimal value of delay that is required; below that minimal value, the system will not be able to operate as desired. More specifically, the current buffer must be processed in the same time it takes to load the next buffer of data, or else data will be lost. For a P4 PC at 1.6 GHz, we found that the minimal delay introduced by the system has been in the order of 150 msec.

We have measured the actual performance of the designed system with an oscilloscope, with the use of an analog signal generator. We generated sinusoidal signals of various frequencies so that we could measure the frequency response of the system. In Fig. 6(a), the results of these measurements are shown for filters 9-11, with frequency ranges 1105-1235 Hz (9<sup>th</sup> filter), 1285-1455 Hz (10<sup>th</sup> filter), 1505-1695 Hz (11<sup>th</sup>

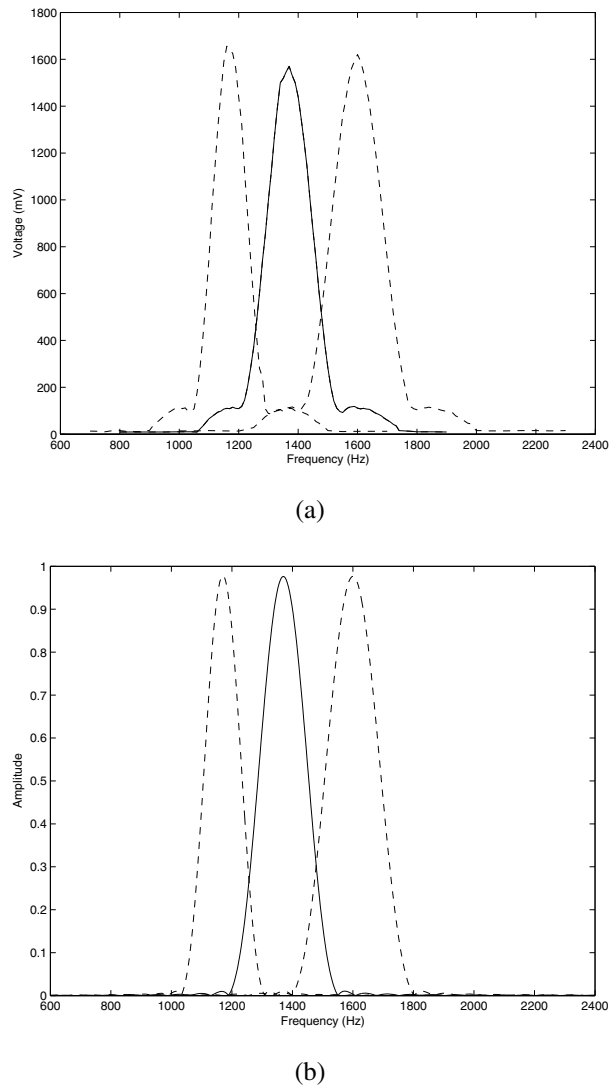


Fig. 6. Three consecutive filters of the sixteen total bandpass filters with frequency ranges (i) 1105-1235 Hz, (ii) 1285-1455 Hz, (iii) 1505-1695 Hz. In (a), the peak-to-peak voltages are depicted as measured with an oscilloscope at the output of the demultiplexer, for a varying frequency sinusoidal input to the system (using an external analog function generator). In (b), the theoretical values of the frequency responses of these filters are shown (same as in Fig. 2).

filter). These results are the output peak-to-peak voltages in mV for all the measured frequencies, for constant sinusoid amplitude. These were measured at the output of the demultiplexer. By comparing this figure with the corresponding theoretically designed filters in Fig. 6(b), we can conclude that the system frequency response is very close to the desired. This was also verified by a closer examination of the measured voltages vs. sinusoid frequency. However, we can see that the sidelobes of the filters are higher than designed (measured around 25 dB lower than the peak value, rather than 40 dB which is the desired). This is due to the temporal overlapping of adjacent channels, that occurred due to practical limitations in the multiplexing/demultiplexing procedure, explained in the following paragraph.

One observation we made during the measurements with the oscilloscope has been that there existed a degree of overlapping between adjacent channels during demultiplexing. This is important since overlapping of different channels in the time-domain will correspond to frequency domain interference of the filter outputs (since each channel corresponds to a particular frequency band). In practice, this overlapping is similar to the sidelobes of one bandpass filter that affect adjacent filters. For the speech recognition algorithm that is implemented in the neural computer, it is important to have clear distinction between all the different frequency bands. Our measurements showed that a filter output can affect the adjacent filter by a "sidelobe" that can reach the level of 25 dB below the desired filter output. However, it must be mentioned that the scenario examined here is the worst case possible. This is true since the measurements were made with a sinusoidal input, which means that at each measurement only one filter will be active, and all others will be zero. Furthermore, the multiplexed signal contains the envelopes for each band, which will be constant for a sinusoid with constant amplitude. Thus, the multiplexed signal will have the form of a square wave, which attains a sinc-like form when passed through the 8 KHz D/A converter of the soundcard. Consequently, the width of the pulse is increased, and inevitably some signal energy enters the adjacent channels. This is worse for the first and last channel, since they are multiplexed adjacently (as can be seen in Fig 5). The consequence for this particular case is that a signal at a very low frequency can produce energy at very high frequencies and vice versa. However, for the case of a real speech signal this effect will be less significant. For a speech signal all frequency bands will be active, and the multiplexed signal will no longer attain a square wave form but that of a continuous signal.

#### IV. CONCLUSIONS

In this paper, we presented a cost-effective implementation for a speech recognition front-end. The objective has been to design a filterbank with a software module, that offers great flexibility for the filter design and for adding any additional functions that might be desired for the front-end. An important restriction to the design of our system has been the fact that the speech recognition task is performed on an analog neural computer, that requires the multiple filtered signals in analog form as its input.

A PC was used for processing using Matlab and its real-time functions were found to be effective for this application. The A/D and D/A conversion tasks were performed by the stereo PC soundcard. Additionally, the neural computer must receive each filter output separately, thus a multiplexing/demultiplexing scheme was needed that was performed in a combined software/hardware implementation. A software module was designed for providing the analog multiplexed signal and the analog synchronization (control) signal. At the same time, we designed a hardware module for demultiplexing the analog multiplexed signal, by first digitizing the analog control signal. All the components used in the hardware part are very inexpensive and easily available. Our measurements showed that the whole system operates successfully. Some overlapping during demultiplexing occurred, however, as explained this was due to the sinusoidal measurement signals used and will be less severe in practical conditions.

#### ACKNOWLEDGEMENTS

This research has been funded by the Catalyst Foundation. In addition, the work of the first author has been partly co-funded by the European Social Fund and National Resources under program EΠΕΑΕΚ Pythagoras.

#### REFERENCES

- [1] P. Mueller, J. Van der Spiegel, D. Blackman, C. Donham, and R. Etienne-Cummings, "A programmable analog neural computer with applications to speech recognition," in *Proc. Comp. & Info. Sc. Symposium (CISS)*, May 1995.
- [2] A. M. A. Ali and J. Van der Spiegel, "Acoustic-phonetic features for the automatic classification of fricatives," *J. Acoust. Soc. Am.*, vol. 109, pp. 2217–2235, May 2001.
- [3] A. M. A. Ali, J. Van der Spiegel, and P. Mueller, "Acoustic-phonetic features for the automatic classification of stop consonants," *IEEE Trans. Speech and Audio Processing*, vol. 9, pp. 833–841, November 2001.
- [4] A. M. A. Ali, J. Van der Spiegel, and P. Mueller, "Robust auditory-based speech processing using the average localized synchrony detector," *IEEE Trans. Speech and Audio Processing*, vol. 10, pp. 279–292, July 2002.
- [5] A. V. Oppenheim and R. W. Schaffer, *Discrete Time Signal Processing*. Prentice Hall, 1989.
- [6] P. Horowitz and W. Hill, *The Art of Electronics*. Cambridge University Press, 2001.